GradLab Submission

Zhigang Wei (zw5259)

Wenqi Yin (wy2985)

Outline

- Brief Intro of Static RRIP Replacement Policy
- Experiment and Quantitative Evaluation
    - Metrics for comparing replacement policy
    - Methodology
    - Result Evaluation
- Result Analysis

Static RRIP-HP Policy

- Uses m-bits per cache block to store one of 2^M Re-Reference Prediction Values (RRPV)
- Each newly inserted block has RRPV of $2^M - 2$
- If hit, the RRPV of that block is set to 0 (HP policy)
- On Replacing, the block of RRPV $2^M - 1$ is replaces with the new block
- If none of the blocks have RRPV of $2^M - 1$, increase each block’s RRPV by 1 until there is one block of RRPV $2^M - 1$

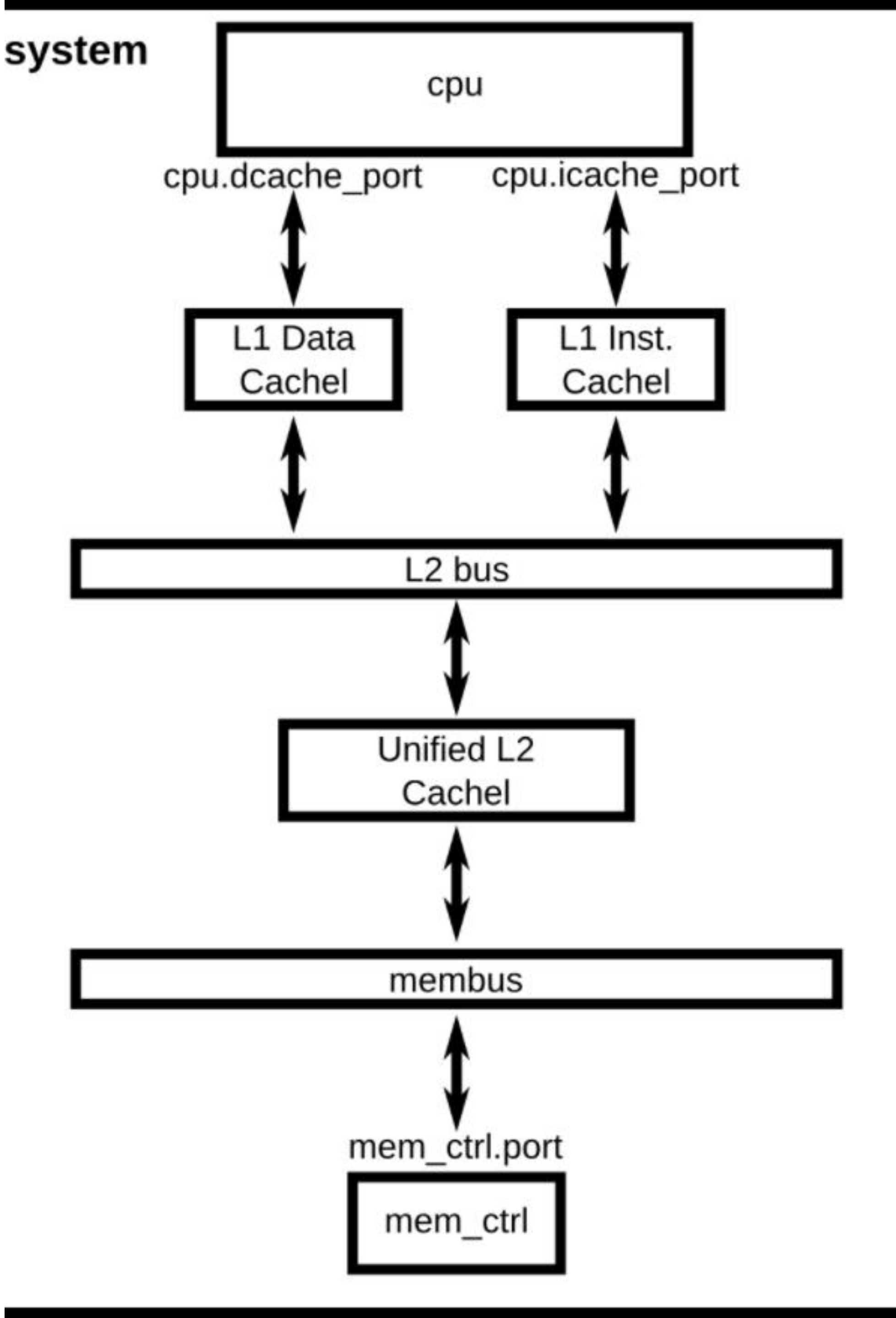
Quantitative Evaluation Metrics

- Miss rate
  - The ideal caches should provide all the data/instructions the processes needed. Thus reduce the amount of time spending on accessing memory

- IPC
  - The aim of cache is to bridge the disparity in the speed of processor and memory system. Thus a better replacement policy means the system can operate more close to the processor time, in other word, a higher IPC

System Configuration

- Per CPU L1 Data Cache and Instruction Cache
    - 2-way set associate
    - 4 MSHRs
    - 32kB icache 64kB dcache
    - LRU replacement policy
- Unified L2 Cache
    - Varies in experiment settings
    - 2MB/8MB cache size
    - 4way/16way set-associate
    - With/without stride prefetcher
    - LRU/Static RRIP replacement policy

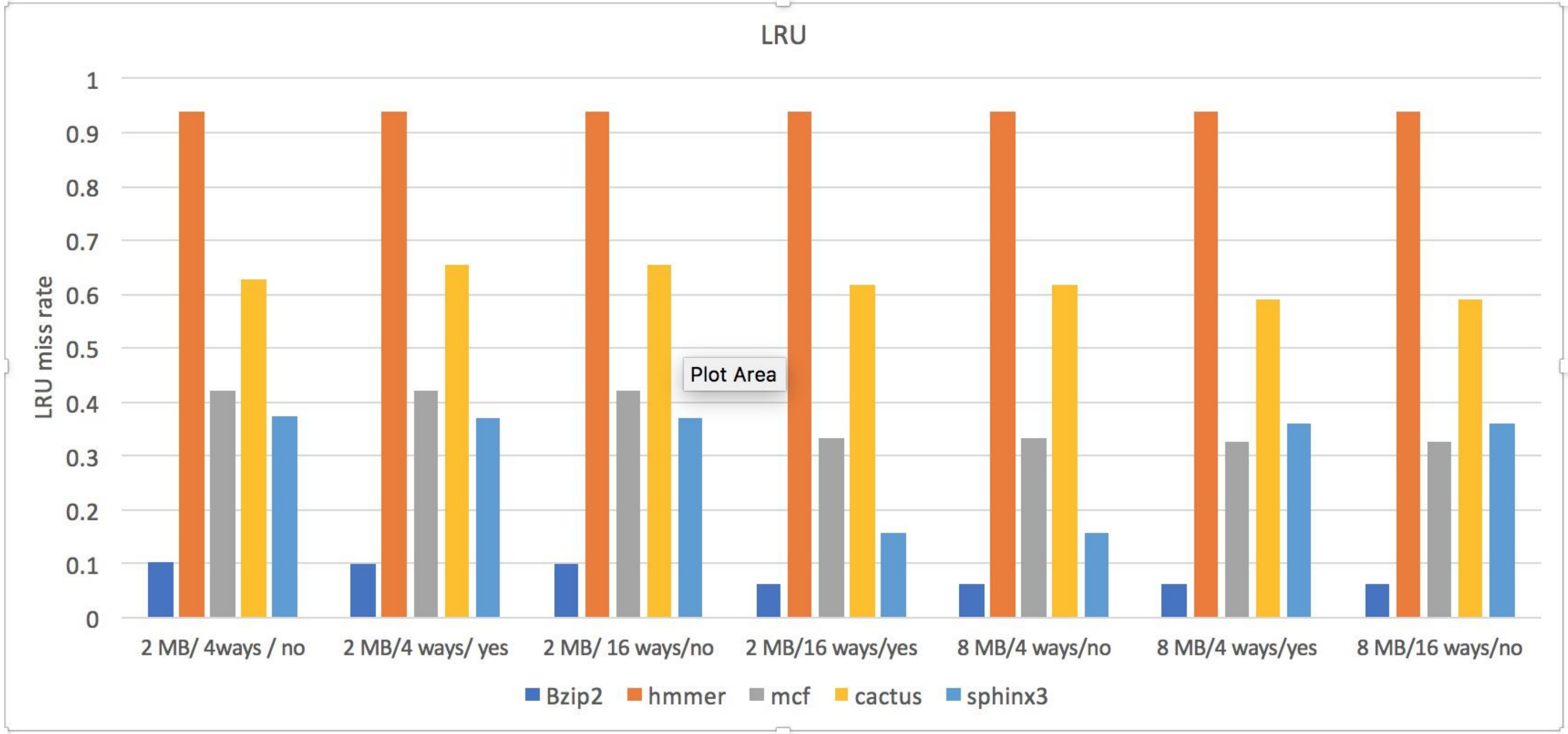


Data Collection

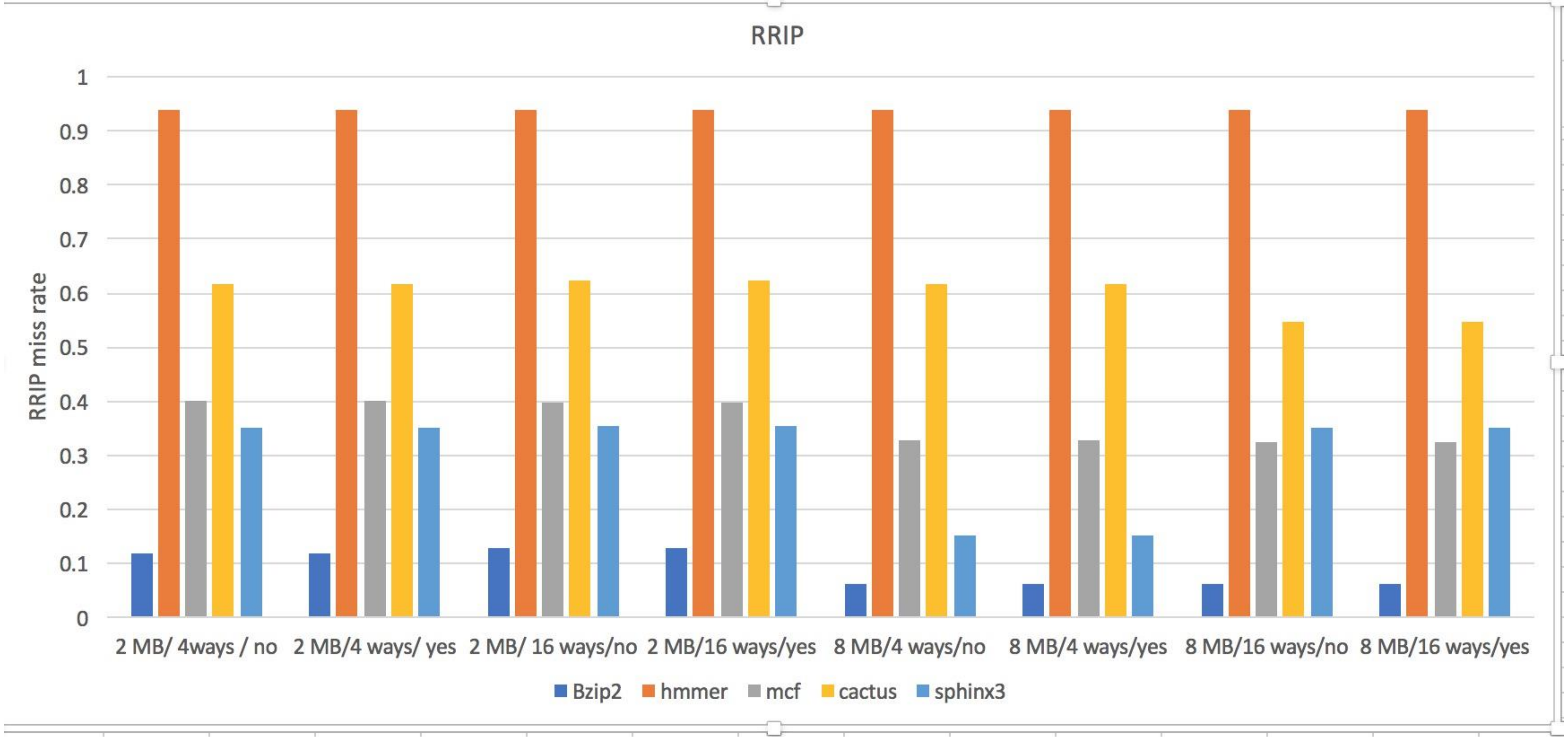
- We experimented on GEM5 simulator, version –u e17949745150
- Choose Testbench in Spec CPU2006
  - Bzip2, cactusADM, hmmer, mcf, sphinx3
- First use valgrind to run the benchmark on native x86 machine and collect profiling data
- Use the profiling data and our uArch simulating instance to generate simpoints and checkpoints
  - For reducing simulation time
- Finally profiling different system configurations on the checkpoints we generated

Quantitative Evaluation

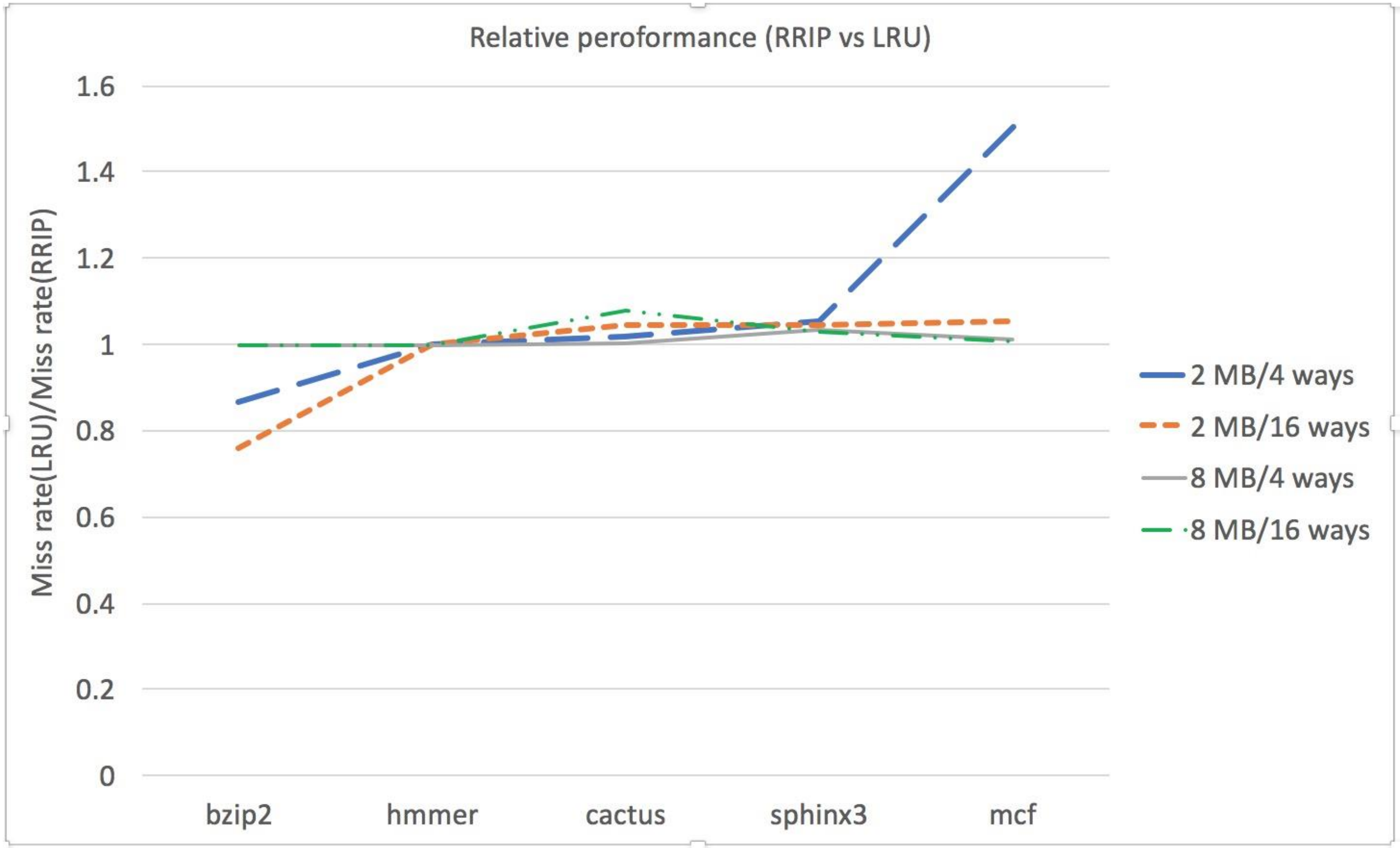
LRU Miss Rate on Various Settings



RRIP Miss-Rate on various Settings



S-Curve


Relative peroformance (RRIP vs LRU)
Miss rate(LRU)/Miss rate(RRIP)
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
bzip2
hmmer
cactus
sphinx3
mcf
2 MB/4 ways
2 MB/16 ways
8 MB/4 ways
8 MB/16 ways

Analysis

- Theoretically, S-RRIP with HP policy records the “usage” of each cacheline and prefers the cachelines which were once hit rather than newly inserted cachelines
- This feature let SRRIP can trace the long-interval locality better than LRU(Because in RRIP, newly inserted cachelines can be evicted quickly, which will never happen in LRU policy)

Analysis Cont’d

- From the experiment result, regardless of the parameters, on most of the benchmarks(cactusADM, sphinx3, hummer, sphinx3) RRIP is slightly better than LRU in terms of miss rate
- This perhaps because in most scenario, RRIP just behave similarly to LRU: The locality over long-interval re-reference happens rarely

Analysis Cont’d

- On bzip2, SRRIP is worse than LRU
- We are not familiar with this compressing algorithm, but suspect that in those algorithms a lot of mid-interval locality presents. Which is not ideal for S-RRIP policy, because S-RRIP will evict those data before they will be re-referenced again
- By saying mid-interval locality, we mean the data accessing pattern which the processes accessing a large arrange of data, while the data got evicted from cache just before they are going to be reused again

Conclusion

- In conclusion, S-RRIPs are ideal for those situations where the workload present a lot of long-interval locality while at the same time the processes also access a wide range of data(without reuse)
- But for the situations where there are more mid-interval locality, SRRIP may perform worse than LRU